Software

REMOVE PYTHON PERFORMANCE BARRIERS FOR MACHINE LEARNING

Anton Malakhov

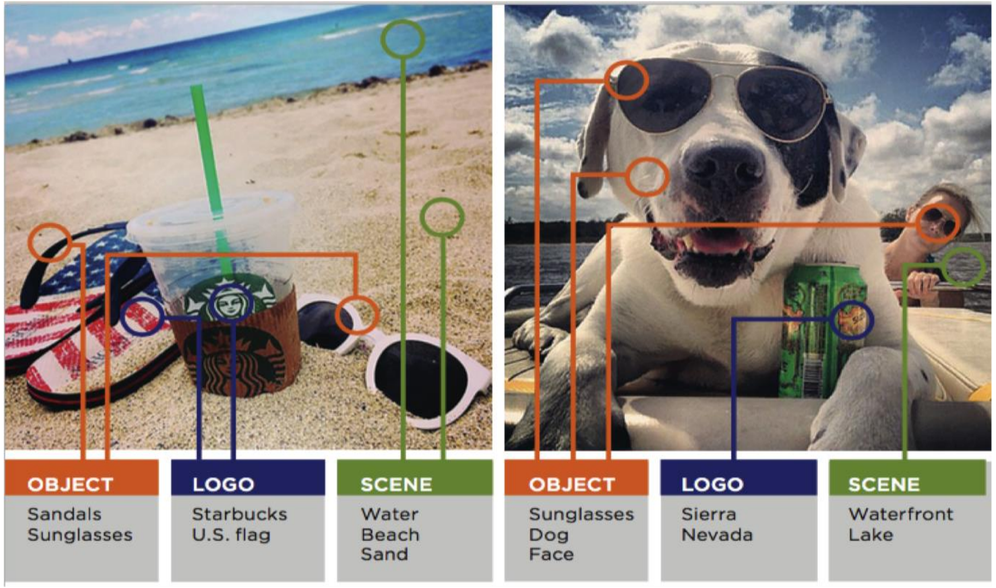
Software Engineer at Intel® Distribution for Python*

Thanks to Sergey Maidanov, Ivan Kuzmin, Oleksandr Pavlyk, Chris Hogan

October 2016

MACHINE LEARNING

- Data is BIG
- Computers are cheap
- How to analyze this data?



Optimization Notice

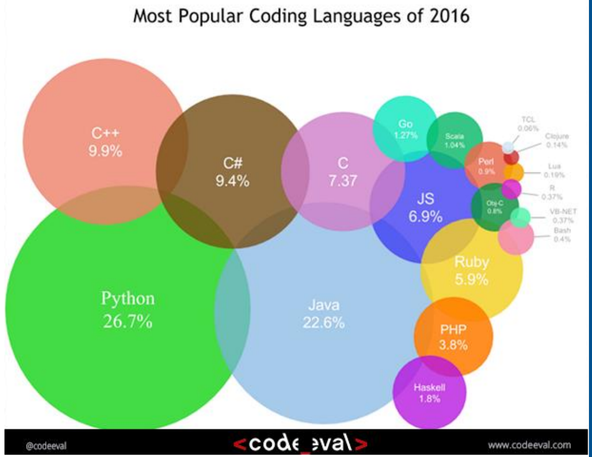
MOTIVATION

Python is among the most popular programming languages

- Especially for prototyping
- But very limited use in production

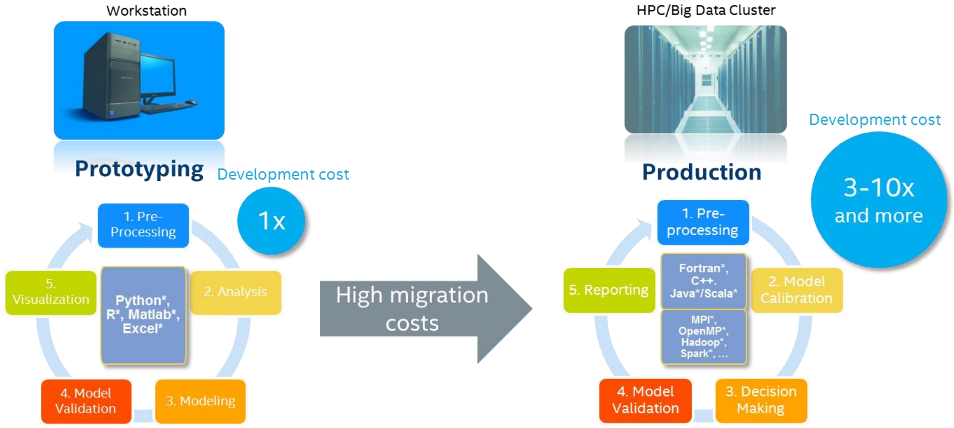
Python is **#1** programming language in **hiring demand** followed by **Java** and **C++**.

And the demand is **growing**

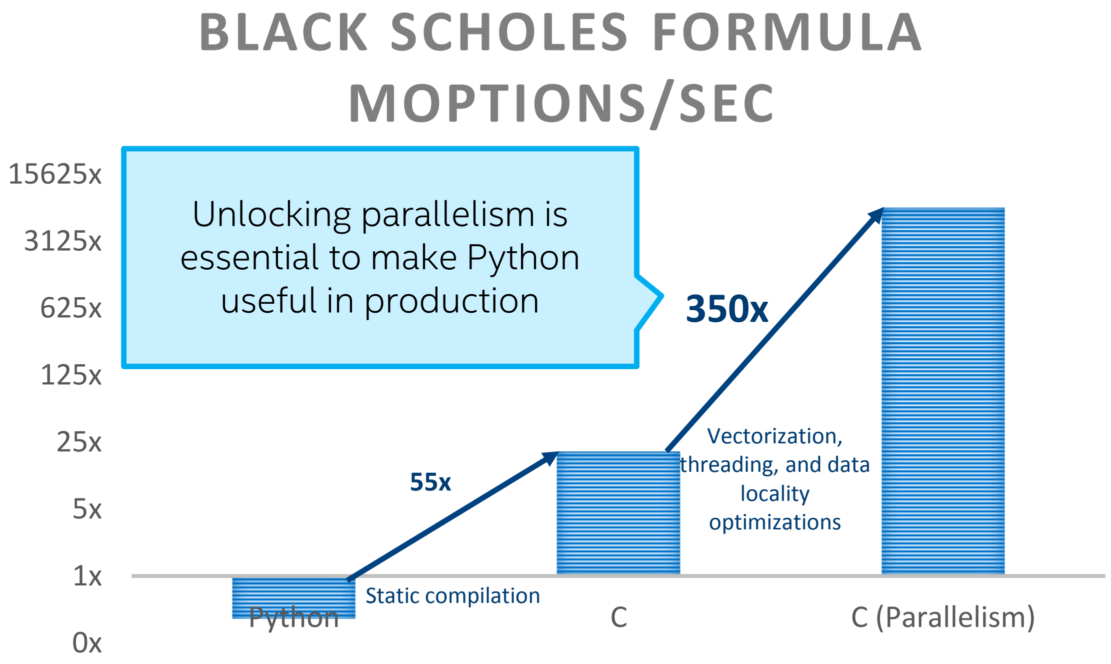


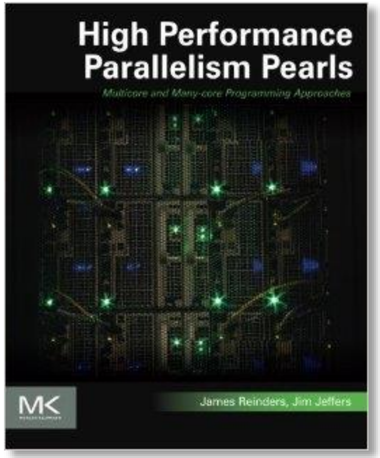
Many computational problems require HPC/Big Data production environments

- Hire a team of Java/C++ programmers … OR
- Ease access for Python researcher and/or have team of Python programmers to deploy optimized Python in production



WHY PARALLELISM MATTERS





Chapter 19. Performance Optimization of Black Scholes Pricing

$$V_{call} = S_0 \cdot \mathrm{CDF}(d_1) - e^{-rT} \cdot X \cdot \mathrm{CDF}(d_2)$$

$$V_{put} = e^{-rT} \cdot X \cdot \mathrm{CDF}(-d_2) - S_0 \cdot \mathrm{CDF}(-d_1)$$

$$d_1 = \frac{\ln\left(\frac{S_0}{X}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{\ln\left(\frac{S_0}{X}\right) + \left(r - \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

PERFORMANCE-PRODUCTIVITY TECHNOLOGICAL OPTIONS

Numerical packages acceleration with Intel® performance libraries (MKL, DAAL, IPP)

Better parallelism and composable multi-threading (OpenMP, TBB, MPI)

Profiling Python and mixed language codes (VTune)

Language extensions for vectorization and multi-threading (Cython, Numba)

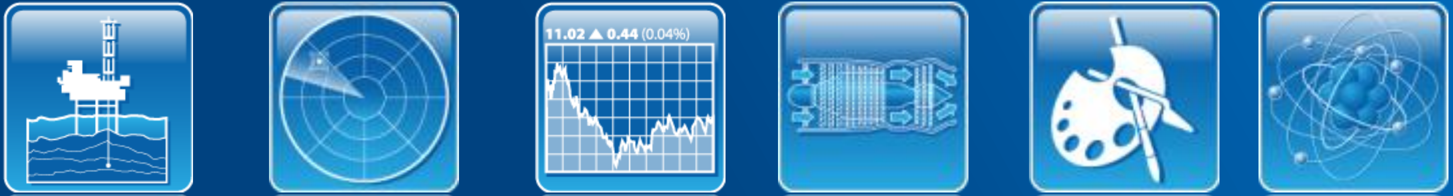
Integration with Big Data and Machine Learning platforms and frameworks (Spark, Hadoop, Theano, etc)

INTEL® DISTRIBUTION FOR PYTHON* 2017

ADVANCING PYTHON PERFORMANCE CLOSER TO NATIVE SPEEDS

Easy, out-of-the-box access to high performance Python	• Prebuilt, optimized for numerical computing, data analytics, HPC • Drop in replacement for your existing Python. No code changes required
Drive performance with multiple optimization techniques	• Accelerated NumPy/SciPy/Scikit-Learn with Intel® Math Kernel Library • Data analytics with pyDAAL, enhanced thread scheduling with TBB, Jupyter* Notebook interface, Numba, Cython • Scale easily with optimized MPI4Py and Jupyter notebooks
Faster access to latest optimizations for Intel architecture	• Distribution and individual optimized packages available through conda and Anaconda Cloud: anaconda.org/intel • Optimizations upstreamed back to main Python trunk

INTEL DISTRIBUTION FOR PYTHON*: NUMERICAL BUILDING BLOCKS



Energy

Signal Processing

Financial Analytics

Engineering Design

Digital Content Creation

Science & Research

NUMPY & SCIPY OPTIMIZATIONS WITH INTEL® MKL

Configuration info: - Versions: Intel® Distribution for Python 2017 Beta, icc 15.0; Hardware: Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz (2 sockets, 16 cores each, HT=OFF), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Operating System: Ubuntu 14.04 LTS.

Linear Algebra

- BLAS
- LAPACK
- ScaLAPACK
- Sparse BLAS
- Sparse Solvers
  - Iterative
  - PARDISO* SMP & Cluster

Up to 100x faster

Fast Fourier Transforms

- Multidimensional
- FFTW interfaces
- Cluster FFT

Up to 10x faster!

Vector Math

- Trigonometric
- Hyperbolic
- Exponential
- Log
- Power
- Root

Up to 10x faster!

Vector RNGs

- Multiple BRNG
- Support methods for independent streams creation
- Support all key probability distributions

Up to 60x faster!

Summary Statistics

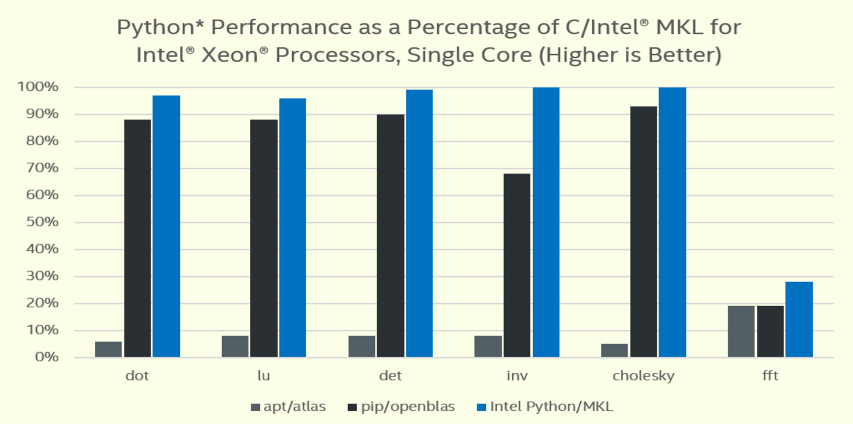
- Kurtosis
- Variation coefficient
- Order statistics
- Min/max
- Variance-covariance

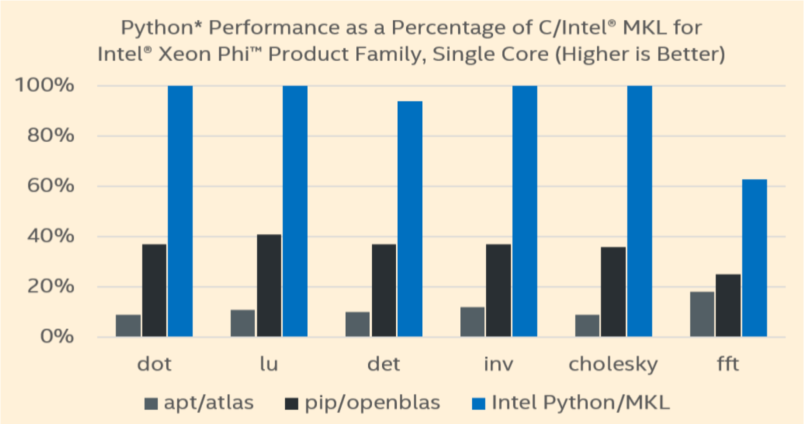
And More

- Splines
- Interpolation
- Trust Region
- Fast Poisson Solver

Functional domain in this color accelerate respective NumPy, SciPy, etc. domain

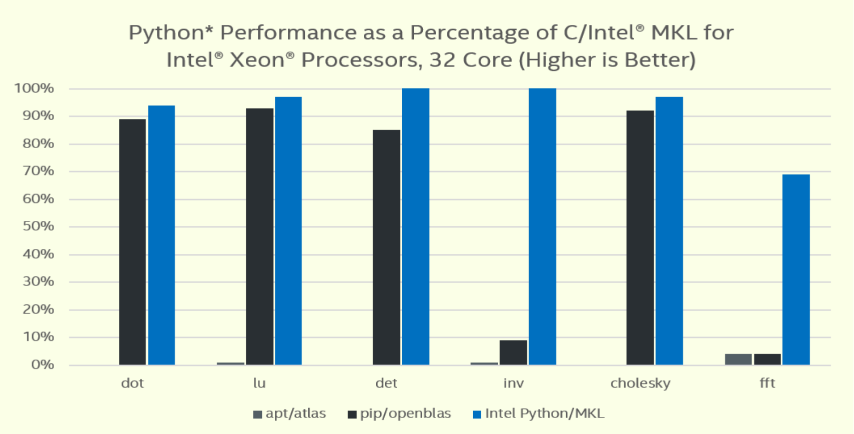
Optimization Notice

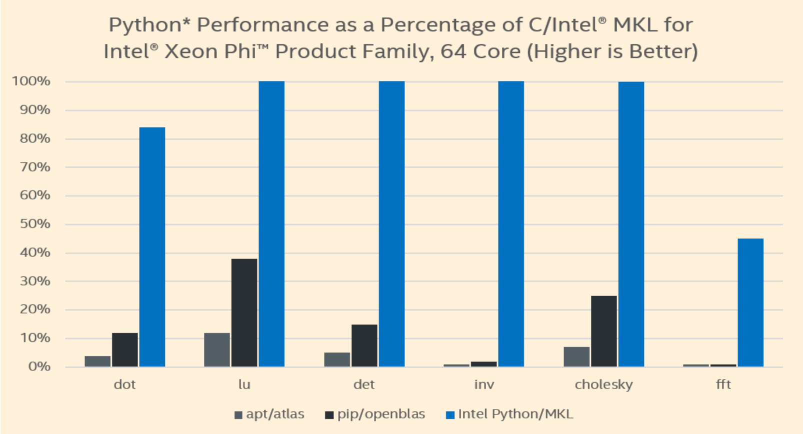




Intel® Xeon® Processor

Intel® Xeon Phi™ Product Family





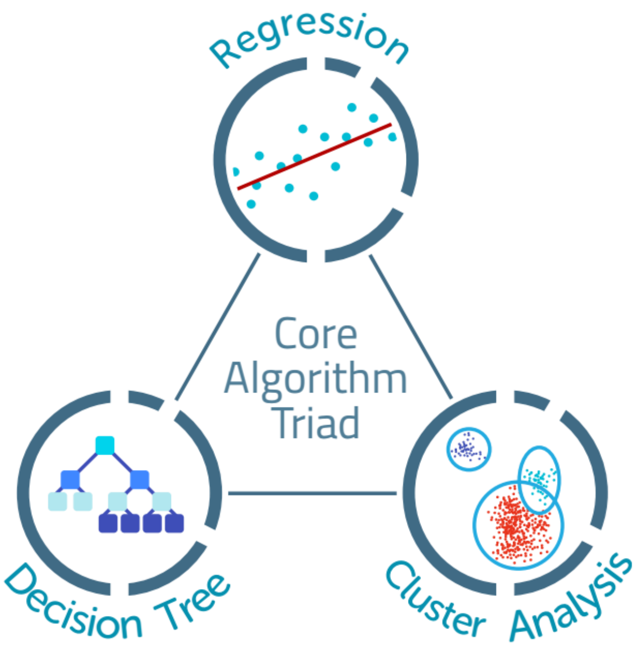
Configuration Info: apt/atlas: installed with apt-get, Ubuntu 16.10, python 3.5.2, numpy 1.11.0, scipy 0.17.0; pip/openblas: installed with pip, Ubuntu 16.10, python 3.5.2, numpy 1.11.1, scipy 0.18.0; Intel Python: Intel Distribution for Python 2017;. Hardware: Xeon: Intel Xeon CPU E5-2698 v3 @ 2.30 GHz (2 sockets, 16 cores each, HT=off), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Xeon Phi: Intel Intel® Xeon Phi™ CPU 7210 1.30 GHz, 96 GB of RAM, 6 DIMMS of 16GB@1200MHz

INTEL DISTRIBUTION FOR PYTHON*: MACHINE LEARNING

MACHINE LEARNING OVERVIEW

What kind of popular algorithms exist in ML:

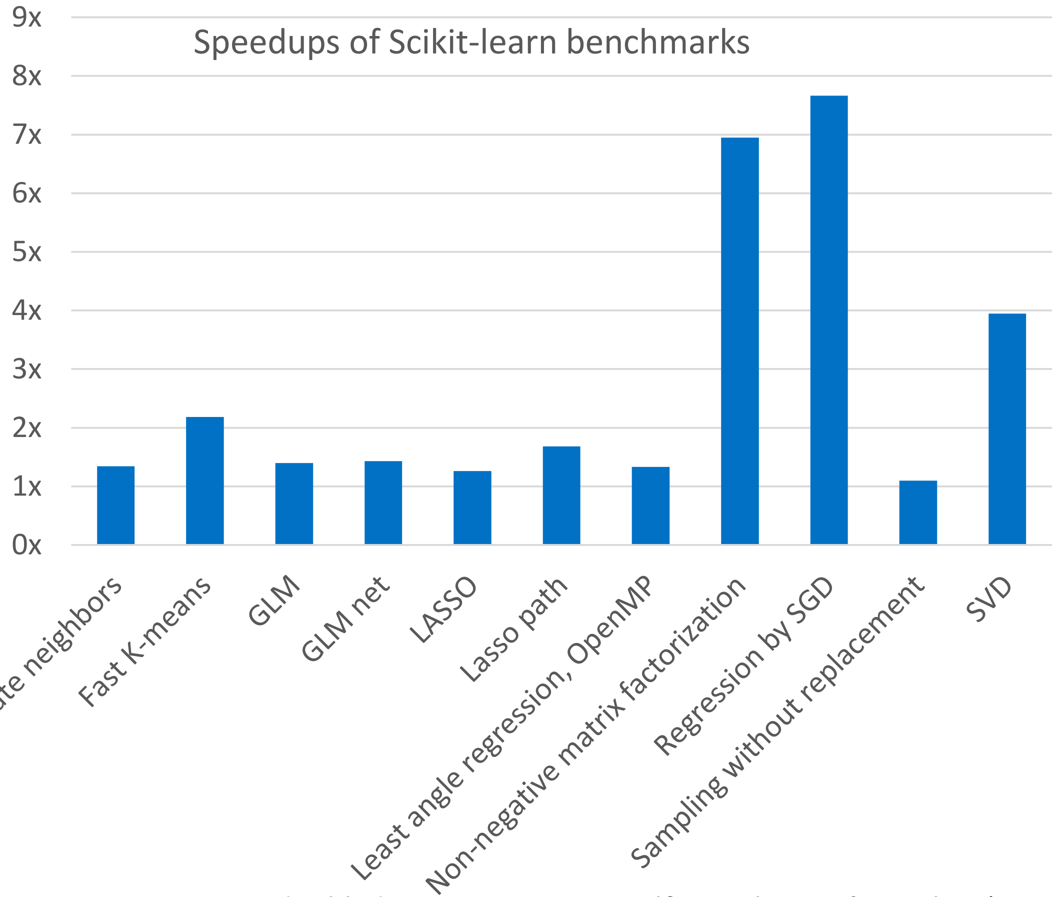
1. Descriptive statistics
   - Moments/correlations/quantiles
   - Including robust methods (for outliers)
2. Factorization/Dimensionality Reductions
   - Find which variables are relevant
3. Clustering
   - Find clusters of data – reduces big data to smaller sizes
4. Regression
   - Find functional relationships in presence of noise
5. Classification
   - Assigning fixed category by features
   - E.g. classification photos by animal/human
   - NN and DL – just part of it



Source: Rexer Analytics report

SCIKIT-LEARN

- Popular machine learning package
- All the popular ML algorithm groups

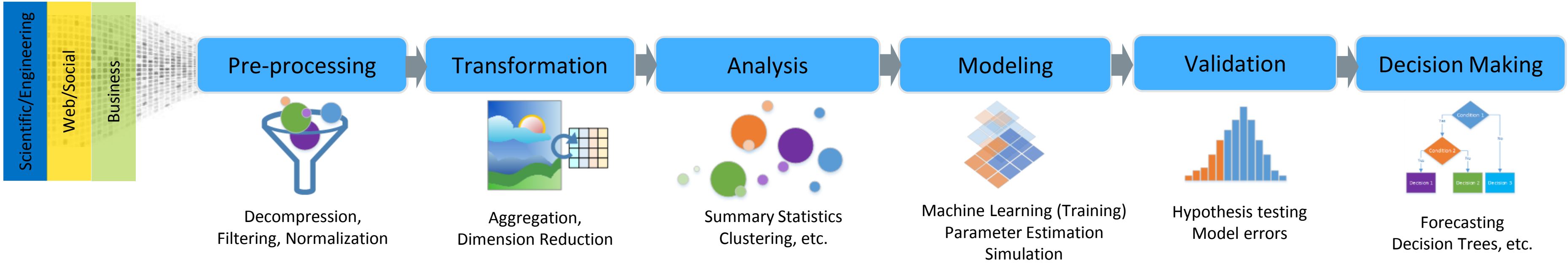


System info: 32x Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz, disabled HT, 64GB RAM; Intel® Distribution for Python* 2017 Gold; Intel® MKL 2017.0.0; Ubuntu 14.04.4 LTS; Numpy 1.11.1; scikit-learn 0.17.1. See Optimization Notice.

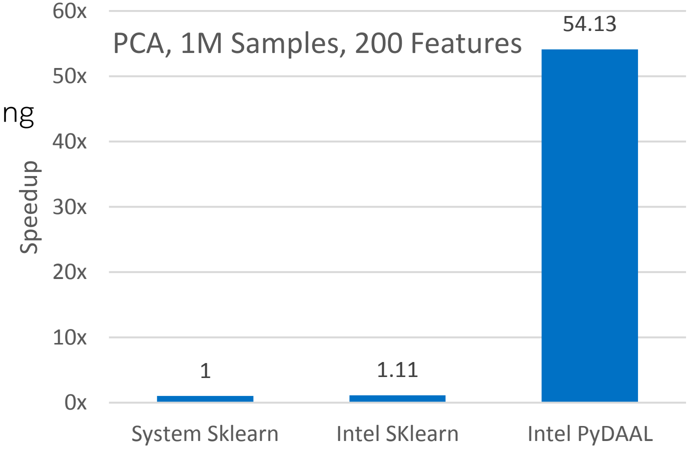
DATA ACQUISITION & TRANSFORMATIONS

- PyTables
  - is a package for managing hierarchical datasets and designed to efficiently and easily cope with extremely large amounts of data.
- Pandas
  - is for data manipulation and analysis; offers data structures and operations for manipulating numerical tables and time series
- DistArray
  - provides general multidimensional NumPy-like distributed arrays
- Dask.DataFrame
  - is a large parallel dataframe composed of many smaller Pandas dataframes, which may live on disk for larger-than-memory computing on a single machine, or in a cluster.

PYDAAL – PYTHON INTERFACES FOR INTEL® DAAL



- pyDAAL delivers significant performance boost
  - Optimizes entire dataflow, from data acquisition to training and prediction
  - Covers different usage scenarios, including online and distributed processing (MPI4PY, PySpark)
- Intel® DAAL available through
  - Intel® Distribution for Python – preinstalled pyDAAL
  - Intel channel at Anaconda.org – pyDAAL package for Conda*
  - Intel® Parallel Studio XE – pyDAAL interface sources for custom package builds
  - OpenDAAL – Github open source project for DAAL, https://software.intel.com/en-us/articles/opendaal

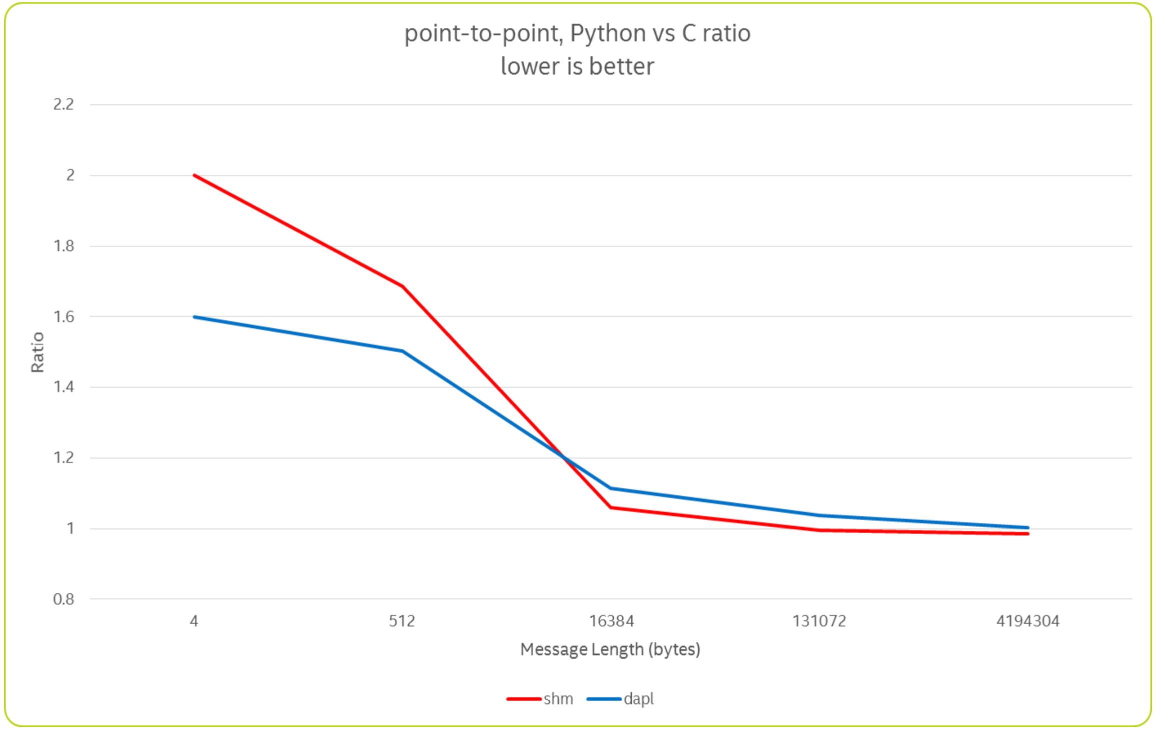


System info: 32x Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz, disabled HT, 64GB RAM; Intel® Distribution for Python* 2017 Gold; Intel® MKL 2017.0.0; Ubuntu 14.04.4 LTS; Numpy 1.11.1; scikit-learn 0.17.1.

INTEL DISTRIBUTION FOR PYTHON*: PARALLELISM

DISTRIBUTED PARALLELISM

- Intel® MPI library
  - Mpi4py
  - ipyparallel
- We also support:
  - PySpark -- Python interfaces for Spark - a fast and general engine for large-scale data processing.
  - Dask -- *a flexible parallel computing library for analytic computing.*



APPLICATION-LEVEL PARALLELISM

- “*speedup is limited by the serial portion of the work*” - Amdahl
  - Python is slow for its serial regions
  - For efficiency, parallelism is needed on application level

- Dask.Array - implements a subset of the NumPy ndarray interface using blocked algorithms, chunking the large array into smaller ones and executing these blocks using multi-threading. Implicit

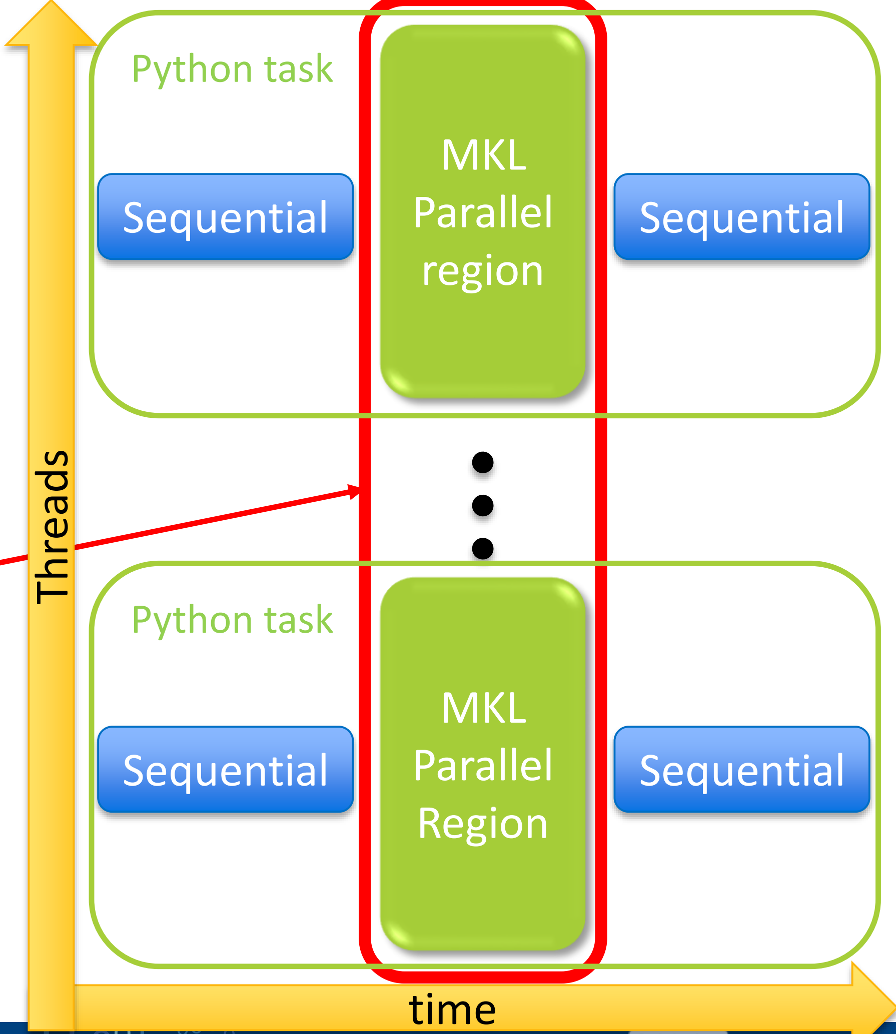
- Joblib, ThreadPool – explicit Python parallelism

- Python’s global lock is not a big issue with native computations

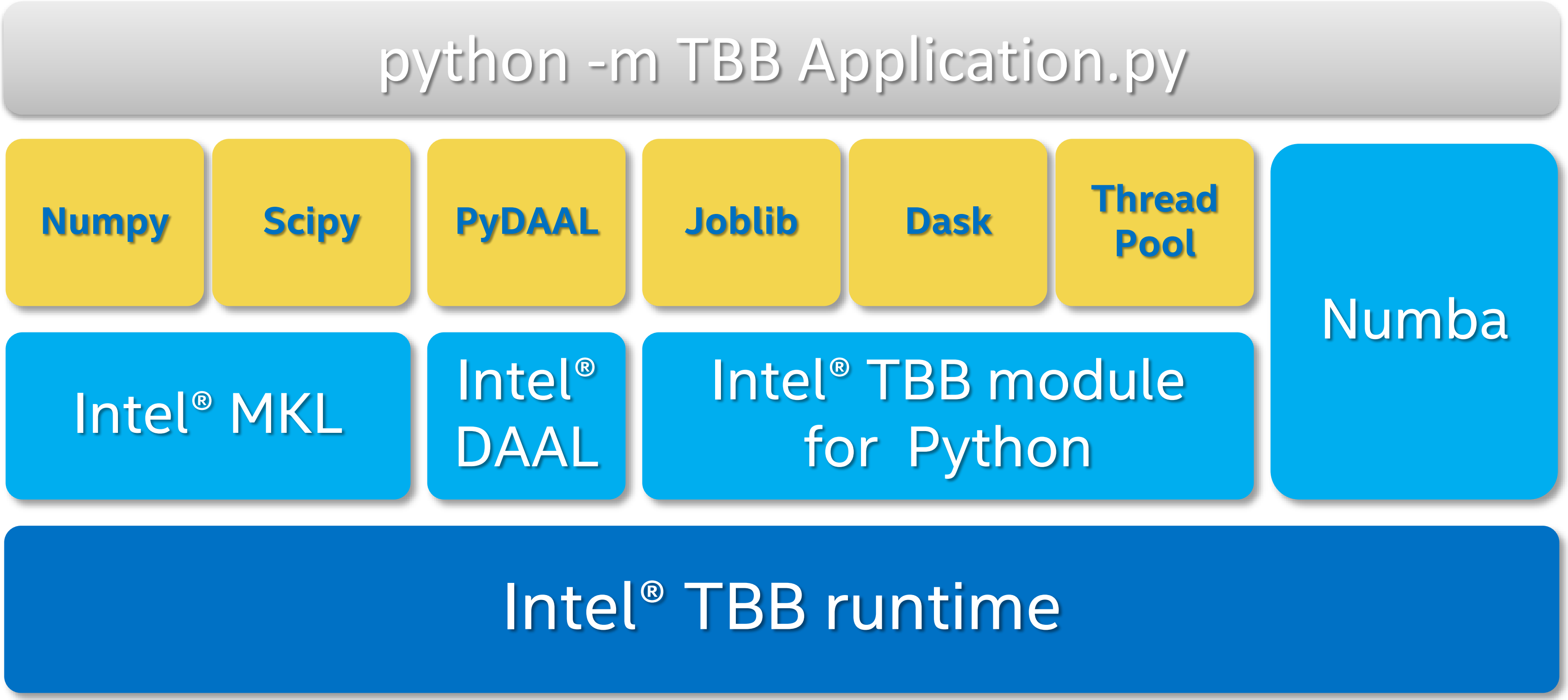
OVER-SUBSCRIPTION ISSUE

- E.g. Dask→Numpy→MKL→OpenMP
- Parallelism on two levels:
  - Dask creates own threads
  - MKL/OpenMP creates threads
- #Software Threads > #HW Threads
  - i.e. parallel regions run in parallel
  - Either performance penalty
  - or fails to create that many threads:

OMP: Error #34: System unable to allocate necessary resources for OMP thread:
OMP: System error #11: Resource temporarily unavailable
OMP: Hint: Try decreasing the value of OMP_NUM_THREADS.



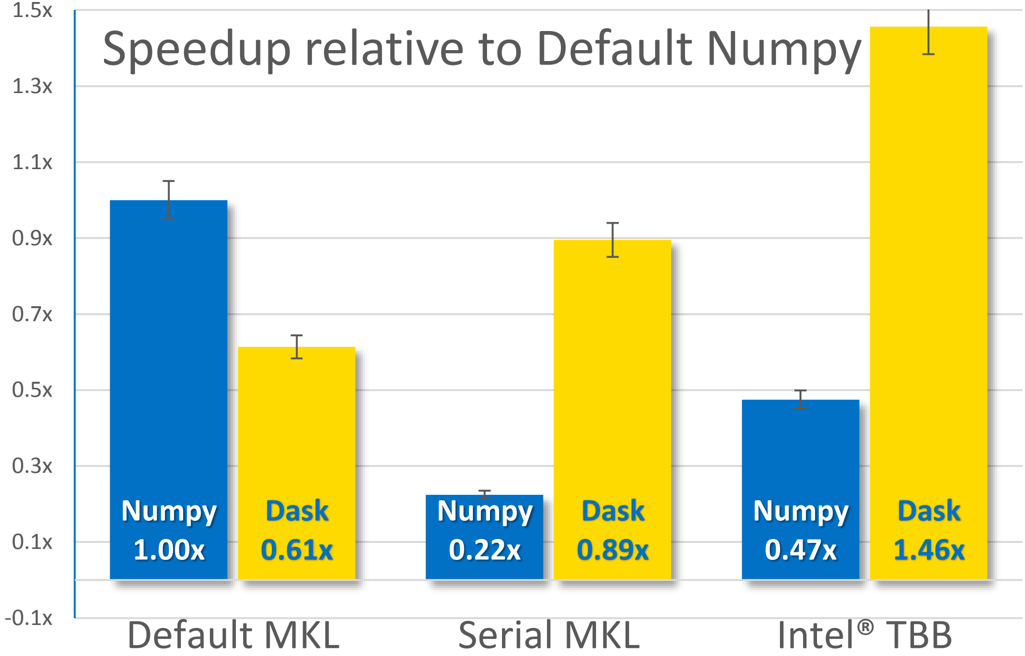
INTEL® TBB: PARALLELISM ORCHESTRATION IN PYTHON ECOSYSTEM



EXAMPLE: QR PERFORMANCE

```python
import time, numpy as np
x = np.random.random((100000, 2000))
t0 = time.time()
q, r = np.linalg.qr(x)
test = np.allclose(x, q.dot(r))
assert(test)
print(time.time() - t0)
```

```python
import time, dask, dask.array as da
x = da.random.random((100000, 2000),
                     chunks=(10000, 2000))
t0 = time.time()
q, r = da.linalg.qr(x)
test = da.all(da.isclose(x, q.dot(r)))
assert(test.compute()) # threaded
print(time.time() - t0)
```

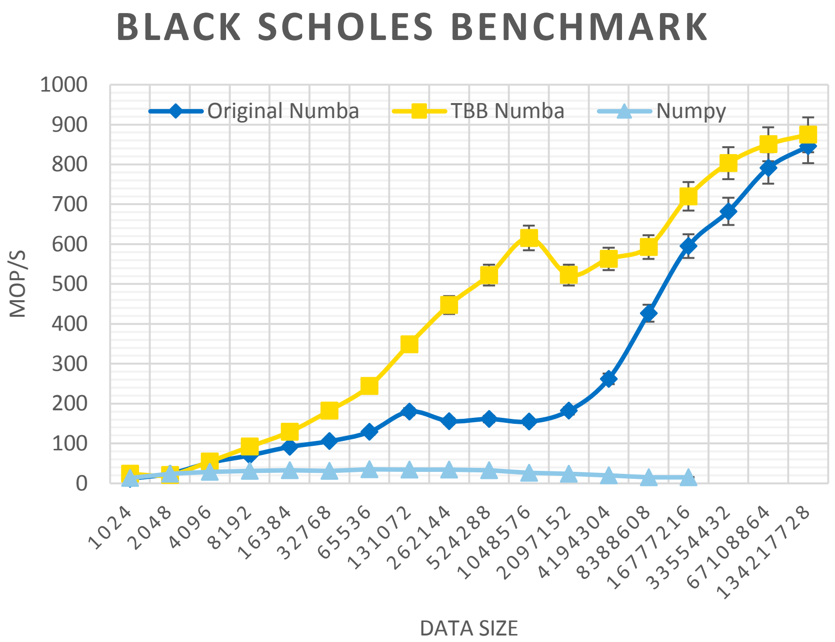


System info: 32x Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz, disabled HT, 64GB RAM; Intel(R) MKL 2017.0 Beta Update 1 Intel(R) 64 architecture, Intel(R) AVX2; Intel(R)TBB 4.4.4; Ubuntu 14.04.4 LTS; Dask 0.10.0; Numpy 1.11.0.

INTEL DISTRIBUTION FOR PYTHON*: PYTHON COMPILERS

NUMBA: JIT COMPILER FOR PYTHON

- With a few annotations, array-oriented and math-heavy Python code can be just-in-time compiled to native machine instructions, similar in performance to C, C++ and Fortran, without having to switch languages or Python interpreters.
- LLVM-based
- Intel optimized with Intel® TBB



Configuration Info: - Versions: Intel(R) Distribution for Python 2.7.11 2017, Beta (Mar 04, 2016), MKL version 11.3.2 for Intel Distribution for Python 2017, Beta, Fedora* built Python*: Python 2.7.10 (default, Sep 8 2015), NumPy 1.9.2, SciPy 0.14.1, multiprocessing 0.70a1 built with gcc 5.1.1; Hardware: 96 CPUs (HT ON), 4 sockets (12 cores/socket), 1 NUMA node, Intel(R) Xeon(R) E5-4657L v2@2.40GHz, RAM 64GB, Operating System: Fedora release 23 (Twenty Three)

CYTHON: COMPILABLE PYTHON

- **Cython** is an **optimising static compiler** for both the Python programming language and the extended Cython programming language (based on **Pyrex**). It makes writing C extensions for Python as easy as Python itself.

- Cython generates C code which can be compiled with Intel C Compiler

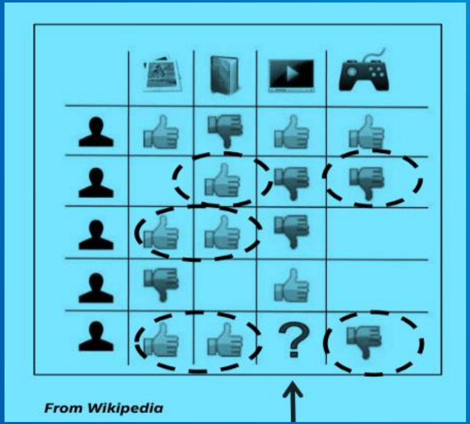
PYTHON PROFILING

INTEL® VTUNE™ AMPLIFIER

- Right tool for high performance application profiling at all levels
  - Function-level and line-level hotspot analysis, down to disassembly
  - Call stack analysis
  - Low overhead
  - Mixed-language, multi-threaded application analysis
  - Advanced hardware event analysis for native codes (Cython, C++, Fortran) for cache misses, branch misprediction, etc.

Feature	cProfile	Line_profiler	Intel® VTune™ Amplifier
Profiling technology	Event	Instrumentation	Sampling, hardware events
Analysis granularity	Function-level	Line-level	Line-level, call stack, time windows, hardware events
Intrusiveness	Medium (1.3-5x)	High (4-10x)	Low (1.05-1.3x)
Mixed language programs	Python	Python	Python, Cython, C++, Fortran

COLLABORATIVE FILTERING CASE STUDY



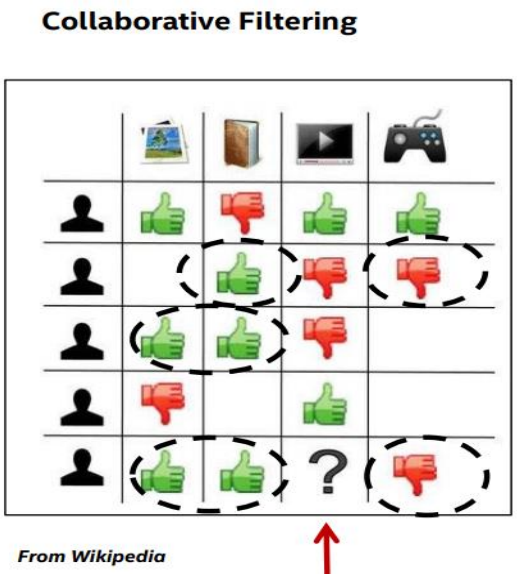
REAL WORLD EXAMPLE

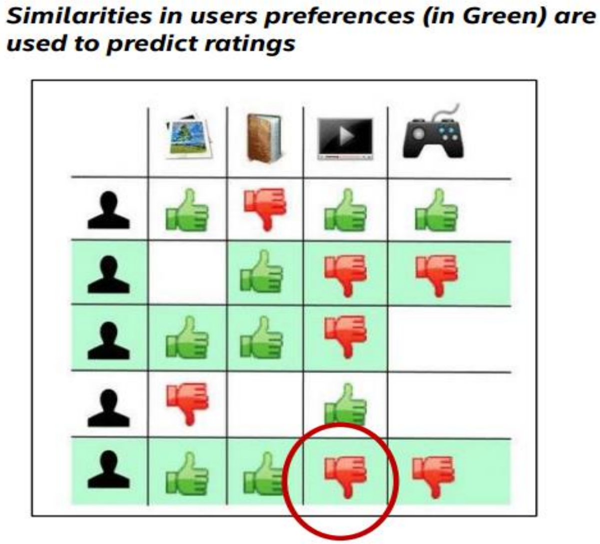
Recommendations of useful purchases

- Amazon, Netflix, Spotify,… use this all the time

COLLABORATION FILTERING

- Processes users' past behavior, their activities and ratings
- Predicts, what user might want to buy depending on his/her preferences





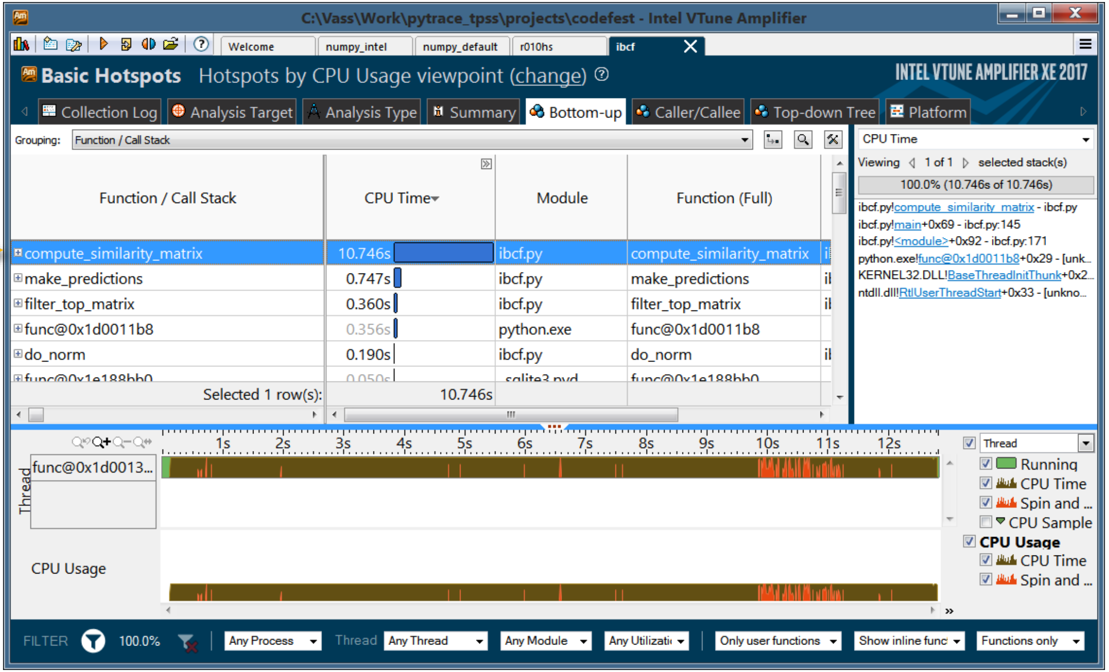
COLLABORATION FILTERING: THE ALGORITHM

- Phase 1: Training
  - Reading of items and its ratings
  - Item-to-item similarity estimation
- Phase 2: Recommendation
  - Reading of user's ratings
  - Generation of recommendations

- Input data was taken from http://grouplens.org/:
  - 1 000 000 ratings.
  - 6040 users
  - 3260 movies

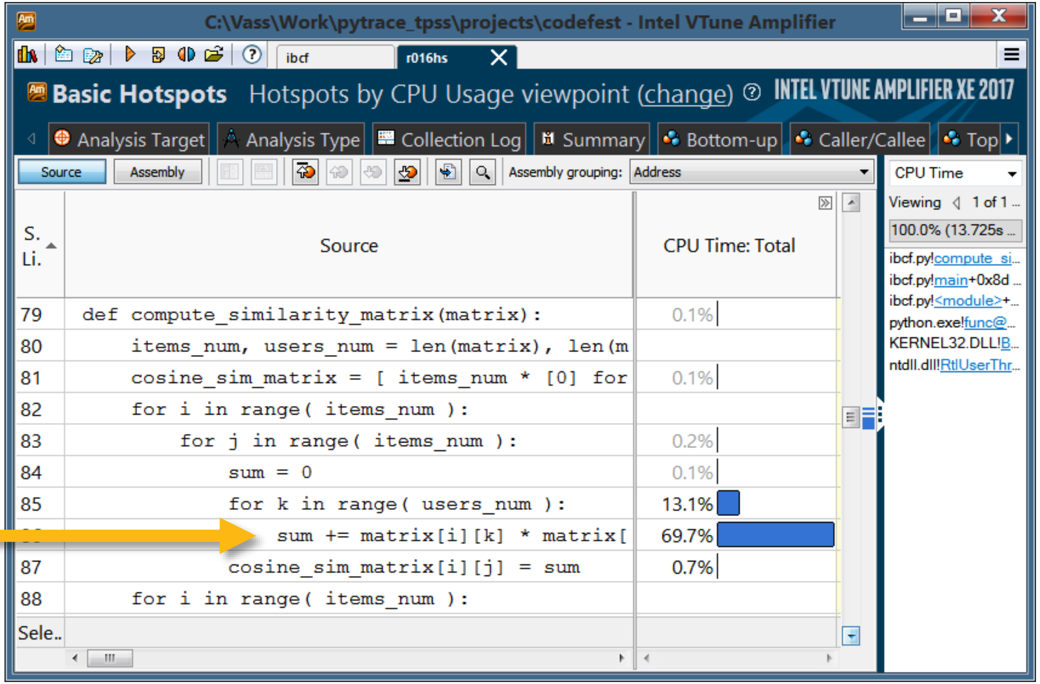
PHASE 1: PROFILING PURE PYTHON COLLABORATIVE FILTERING

Items similarity assessment (similarity matrix computation) is the main hotspot



Configuration Info: - Versions: Red Hat Enterprise Linux* built Python*: Python 2.7.5 (default, Feb 11 2014), NumPy 1.7.1, SciPy 0.12.1, multiprocessing 0.70a1 built with gcc 4.8.2; Hardware: 24 CPUs (HT ON), 2 Sockets (6 cores/socket), 2 NUMA nodes, Intel(R) Xeon(R) X5680@3.33GHz, RAM 24GB, Operating System: Red Hat Enterprise Linux Server release 7.0 (Maipo)

PHASE 1: PROFILING PURE PYTHON COLLABORATIVE FILTERING

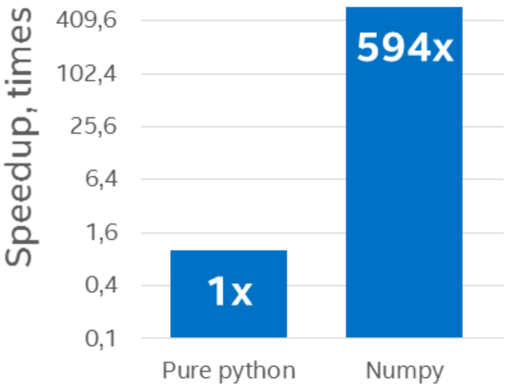


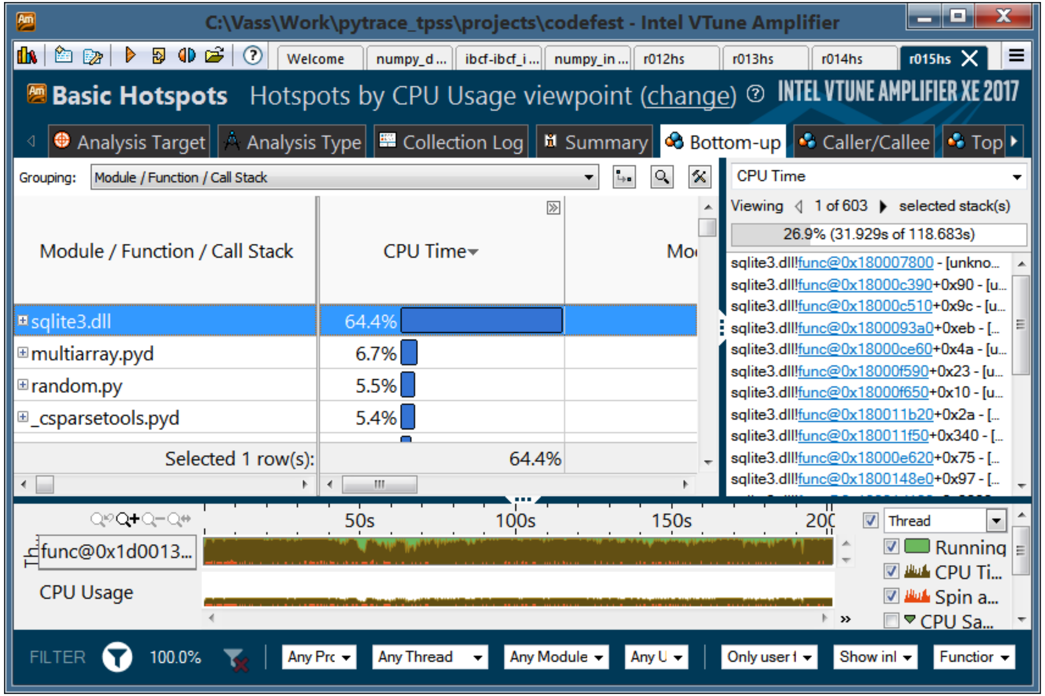
This loop is major bottleneck. Use appropriate technologies (NumPy/SciPy/Scikit-Learn or Cython/Numba) to accelerate

Configuration Info: - Versions: Red Hat Enterprise Linux* built Python*: Python 2.7.5 (default, Feb 11 2014), NumPy 1.7.1, SciPy 0.12.1, multiprocessing 0.70a1 built with gcc 4.8.2; Hardware: 24 CPUs (HT ON), 2 Sockets (6 cores/socket), 2 NUMA nodes, Intel(R) Xeon(R) X5680@3.33GHz, RAM 24GB, Operating System: Red Hat Enterprise Linux Server release 7.0 (Maipo)

PHASE 1: PYTHON + NUMPY (MKL)

- Much faster!
- The most compute-intensive part takes ~5% of all the execution time





Configuration info: 96 CPUs (HT ON), 4 Sockets (12 cores/socket), 1 NUMA nodes, Intel(R) Xeon(R) E5-4657L v2@2.40GHz, RAM 64GB, Operating System: Fedora release 23 (Twenty Three)

Optimization Notice

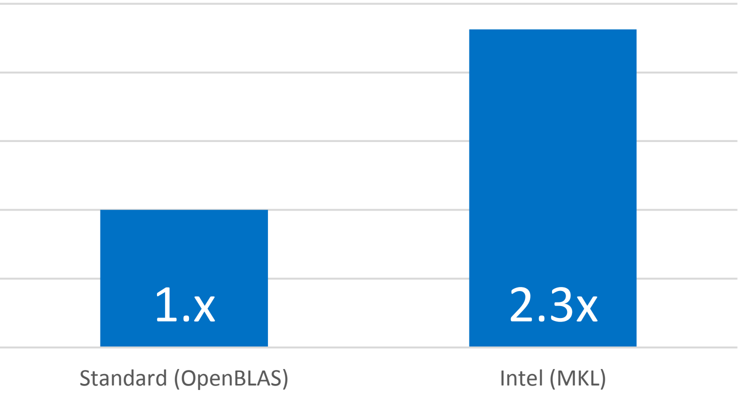
PHASE 2: GENERATION OF USER RECOMMENDATIONS

```
# Define custom compute step
@numba.guvectorize('(f8[:],f8[:])', '(),()',
                   target="parallel")
def masking(x, rating):
    if rating[0]:
        x[0] = 0

# Numpy arrays for 3260 items and 500K users
topk_matrix =numpy.empty((3260,  3260),dtype='f8')
user_ratings=numpy.empty((3260,500000),dtype='f8')

# Compute recommendation
x = topk_matrix.dot(user_ratings) # call Numpy
masking(x, user_ratings)          # call Numba
recommendation_ids = x.argmax(axis=0)
```

User requests per second, Intel® Xeon Phi™



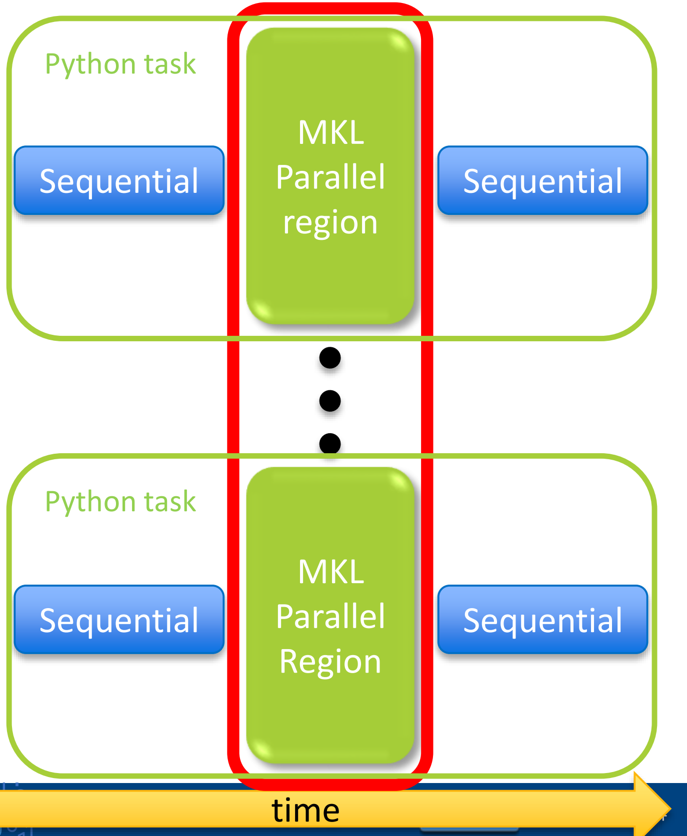
Configuration Info: Hardware: Intel® Xeon Phi™ CPU 7250, 68 cores @1.40GHz, 96GB DDR4-2400
Versions: Intel® Distribution for Python 2017 Gold, Intel® MKL version 2017.0.0, libopenblasp-r0-39a31c03.2.18.so, Python 3.5.2, NumPy 1.11.1, SciPy 0.18.0; Red Hat Enterprise Linux Server 7.2

PHASE 2: EVEN FASTER WITH DASK, MULTI-THREADED APPLICATION

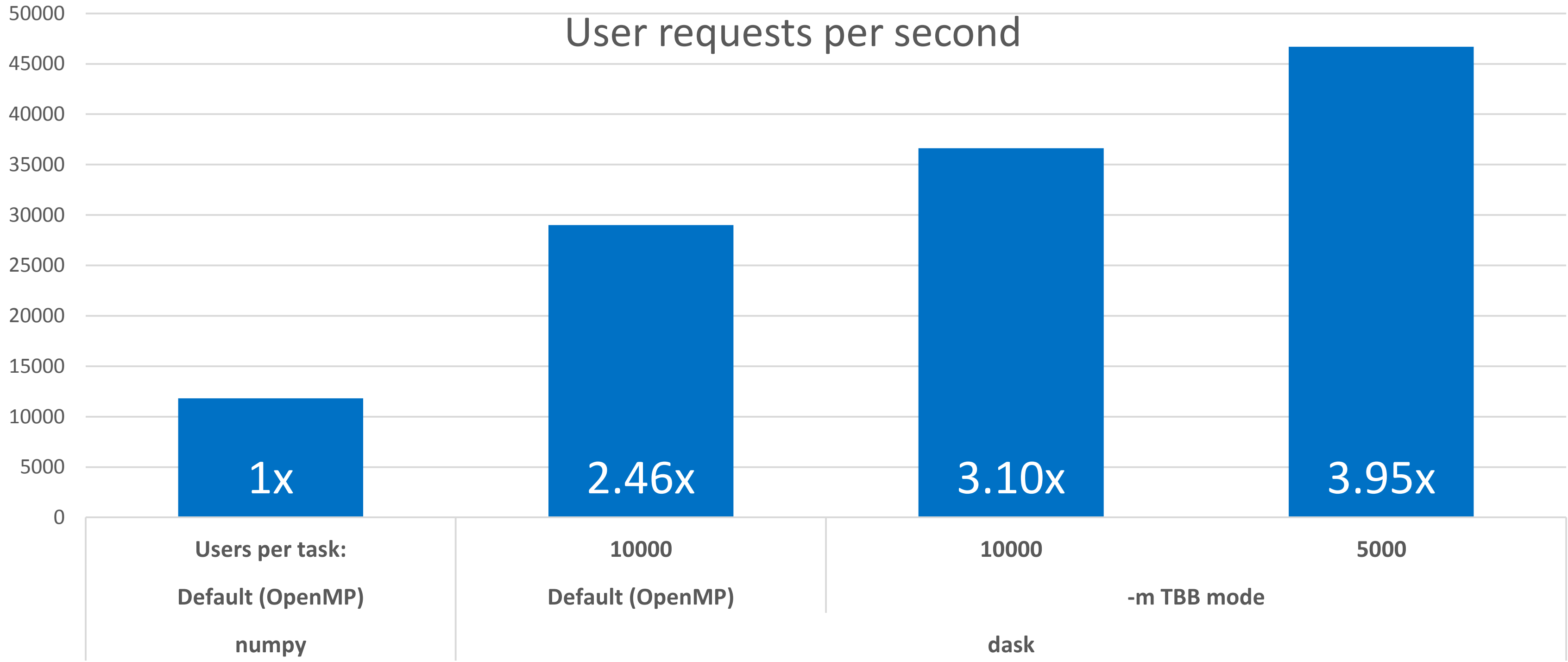
```
# Define custom compute step
@numba.guvectorize('(f8[:],f8[:])', '(),()',
                   target="parallel")
def masking(x, rating):
    if rating[0]:
        x[0] = 0

# use dask.array instead of numpy array
chunks = (3260, 5000) # 5000 users per task here
topk_matrix = dask.array.empty((3260,  3260), chunks)
user_ratings = dask.array.empty((3260, 500000), chunks)

# Dask array program is like Numpy but multi-threaded
x = topk_matrix.dot(user_ratings)
dask.array.map_blocks(masking, x, user_ratings)
recommendation_ids = x.argmax(axis=0).compute()
```



PHASE 2: MORE PERFORMANCE WITH NESTED PARALLELISM



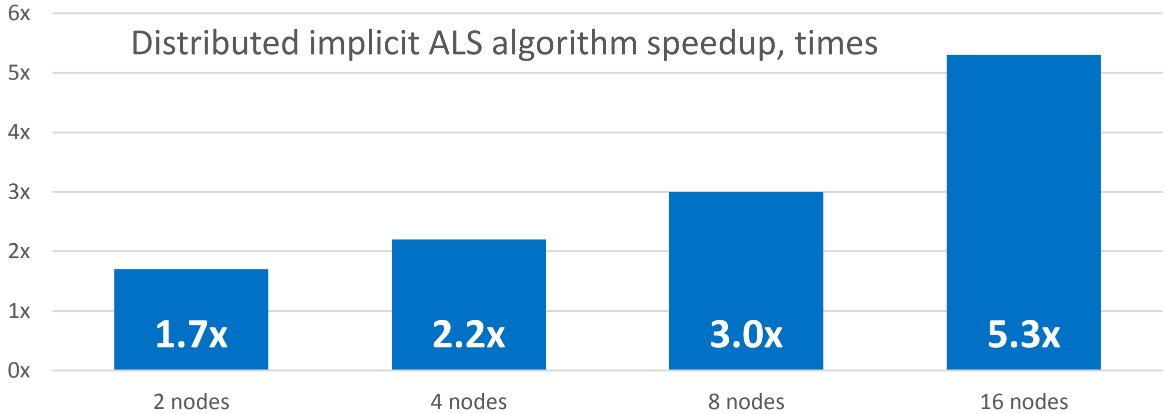
Hardware: Intel® Xeon® CPU E7-8890 v4, 4x24 cores @ 2.20GHz (4GHz max), HT is OFF, 768 GB DDR4; Versions: Intel® Distribution for Python* 2017 Gold, Intel® MKL version 2017.0.0, Python 3.5.2, NumPy 1.11.1, SciPy 0.18.0, Numba 0.26.0, llvmlite 0.11.0, Dask 0.11.0, CentOS Linux release 7.2.1511 (Core).

MORE REALISTIC APPROACH: DISTRIBUTED COLLABORATIVE FILTERING

- Big Data doesn’t fit one node efficiently
- Distributed algorithms are hard to implement
- Using out-of-the-box PyDAAL algorithm instead

PHASE 1: COLLABORATIVE FILTERING WITH PYDAAL AND MPI4PY

- PyDAAL implements Implicit Alternating Least Squares algorithm
  - Single node and distributed variant
  - Handles sparse and dense datasets
  - See code samples for details https://software.intel.com/en-us/intel-daal-support/code-samples



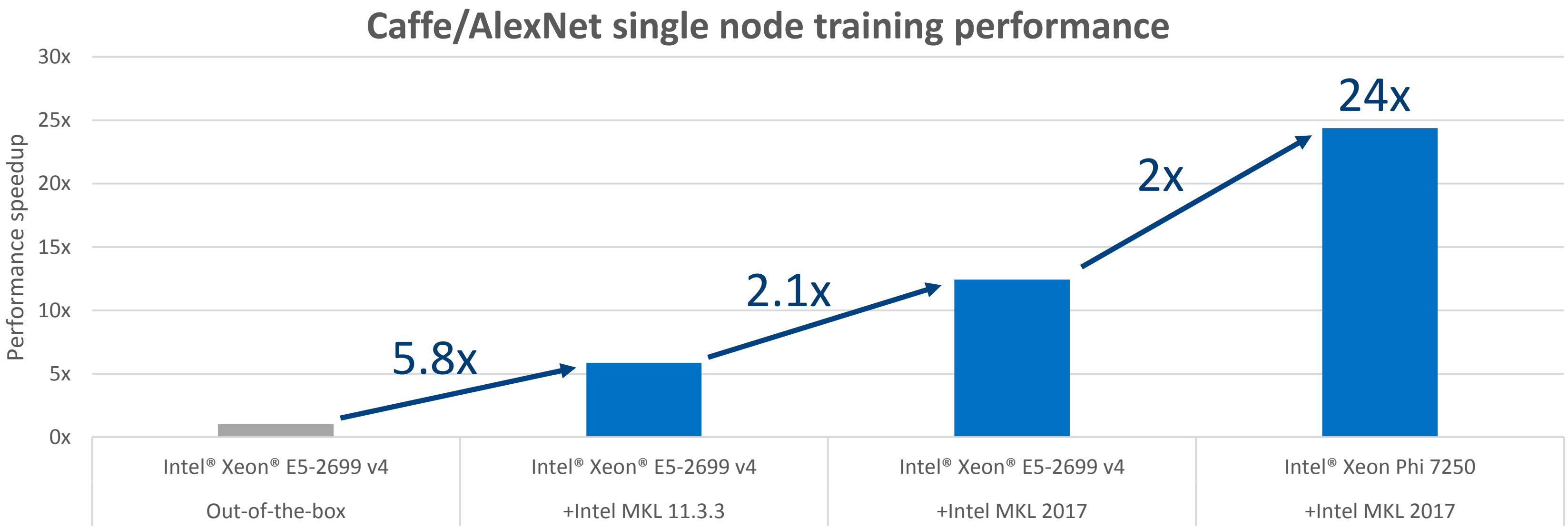
Configuration Info:
Hardware (each node): Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz, 2x18 cores, HT is ON, RAM 128GB;
Versions: Oracle Linux Server 6.6, Intel® DAAL 2017 Gold, Intel® MPI 5.1.3;
Interconnect: 1 GB Ethernet

WORK IN PROGRESS

INTEL IS WORKING ON IA OPTIMIZATION FOR DEEP LEARNING

- Theano
  - https://github.com/intel/theano Intel fork
- Caffe
  - https://github.com/intel/caffe Intel fork
- TensorFlow
  - works great with Intel® Distribution for Python*
- Neon
  - Intel acquired Nervana Systems: https://www.nervanasys.com/intel-nervana/

CAFFE ACCELERATED POWERED BY INTEL® MKL (TRAINING)



Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance . *Other names and brands may be property of others

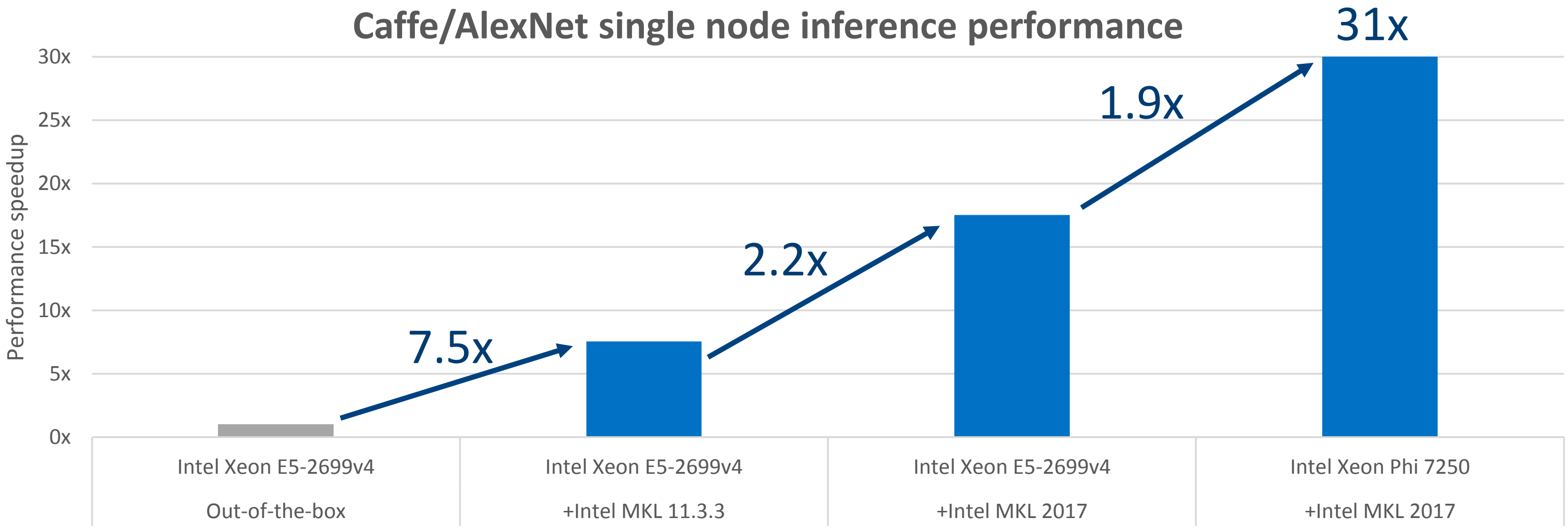
Configurations:

- 2 socket system with Intel® Xeon Processor E5-2699 v4 (22 Cores, 2.2 GHz,), 128 GB memory, Red Hat* Enterprise Linux 6.7, BVLC Caffe, Intel Optimized Caffe framework, Intel® MKL 11.3.3, Intel® MKL 2017
- Intel® Xeon Phi™ Processor 7250 (68 Cores, 1.4 GHz, 16GB MCDRAM), 128 GB memory, Red Hat* Enterprise Linux 6.7, Intel® Optimized Caffe framework, Intel® MKL 2017

All numbers measured without taking data manipulation into account.

Optimization Notice

CAFFE ACCELERATED POWERED BY INTEL® MKL (INFERENCE)

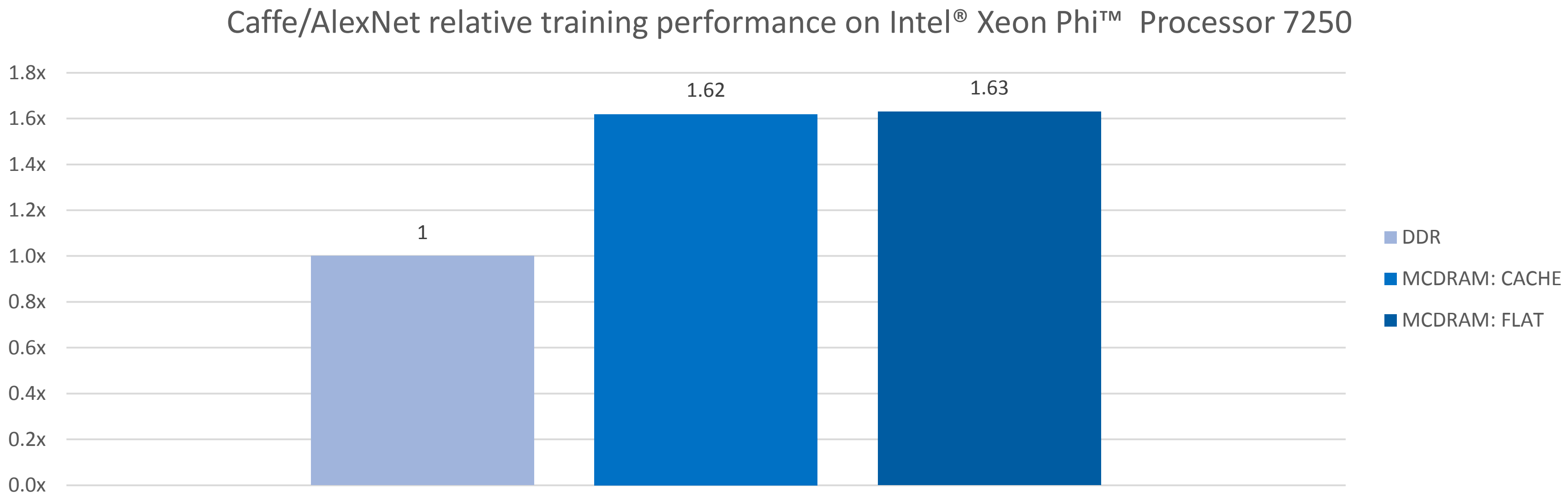


Configurations:

- 2 socket system with Intel® Xeon® Processor E5-2699 v4 (22 Cores, 2.2 GHz,), 128 GB memory, Red Hat* Enterprise Linux 6.7, BVLC Caffe, Intel Optimized Caffe framework, Intel® MKL 11.3.3, Intel® MKL 2017
- Intel® Xeon Phi™ Processor 7250 (68 Cores, 1.4 GHz, 16GB MCDRAM), 128 GB memory, Red Hat* Enterprise Linux 6.7, Intel® Optimized Caffe framework, Intel® MKL 2017

All numbers measured without taking data manipulation into account.

BETTER PERFORMANCE IN DEEP NEURAL NETWORK WORKLOADS WITH MCDRAM (SPECIAL MEMORY)



Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance . *Other names and brands may be property of others

Configurations:

- Intel® Xeon Phi™ Processor 7250 (68 Cores, 1.4 GHz, 16GB MCDRAM), 128 GB memory, Red Hat* Enterprise Linux 6.7, Intel® Optimized Caffe framework, Intel® MKL 2017 Beta Update 1

All numbers measured without taking data manipulation into account.

Optimization Notice

CALL FOR ACTION

"I expected Intel's numpy to be fast but it is significant that plain old python code is much faster with the Intel version too."

Dr. Donald Kinghorn, Puget Systems Review

InfoWorld **Intel's Python distribution turbocharges data science**

Intel Distribution for Python adds Intel's high-speed math libraries to the existing, highly convenient Anaconda version for data scientists

- Download and use it! It's free
  - https://software.intel.com/python-distribution
- Easy to install with Anaconda
  - https://anaconda.org/intel/
- Commercial support via Intel® Parallel Studio 2017

insideHPC **HPC Podcast Looks at Intel's Pending Distribution of Python**

Yes, Intel is doing their own Python build! It is still in beta but I think it's a great idea. ……….Yeah, it's important!

LEGAL DISCLAIMER & OPTIMIZATION NOTICE

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

For more complete information about compiler optimizations, see our Optimization Notice at https://software.intel.com/en-us/articles/optimization-notice#opt-en.

intel®